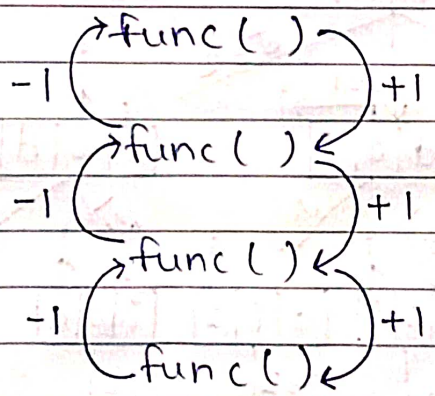
20/10/2023
Friday

Backtracking,

Backtracking is nothing but recursion. Recursive call krte time jo additional actions perform kiye the, recursive call se return krte time vo saare actions ko nullify krna hai.
It's a technique jisme problem statement ke saare solutions explore krte hai keeping in mind don't explore any case again.

For instance,



Every recursive call adds 1 to the previous result. So, to nullify this operation, we are subtracting 1 when returning.

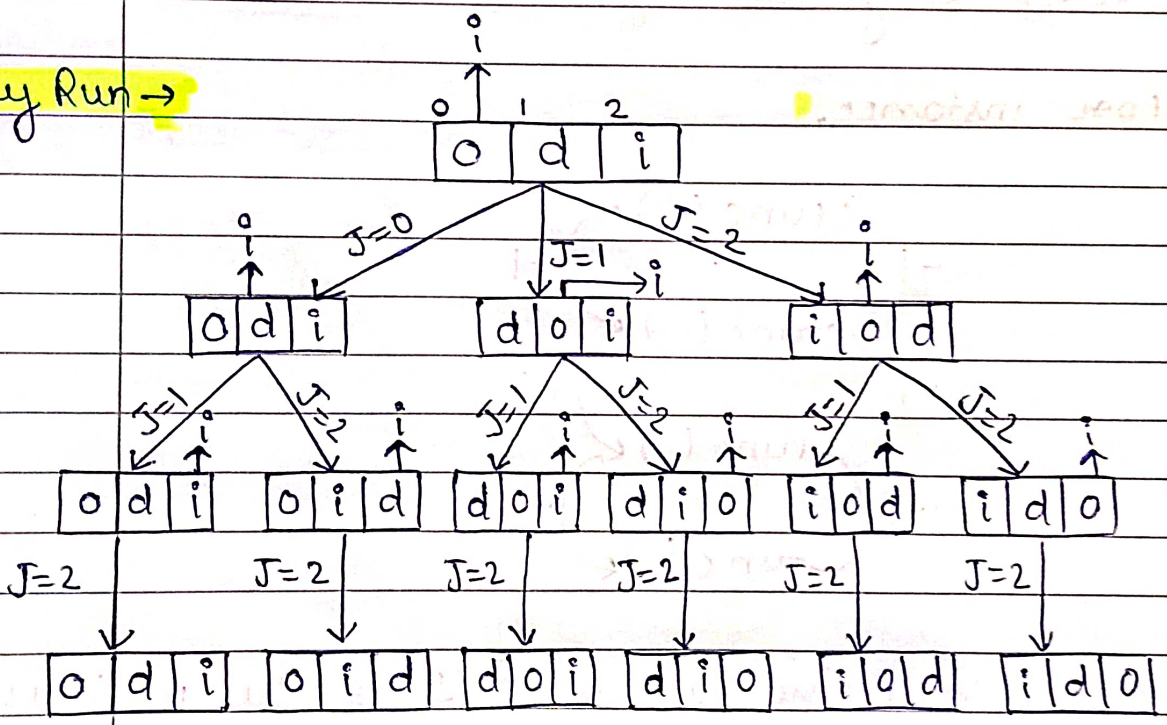
* **Permutations of string →**

I/P → str = "odi"

O/P →

odi
oid
doi
dio
ido
iod

Dry Run →



Code →

```
#include <bits/stdc++.h>
using namespace std;

void stringPermutation (string &str, int index){

    // base case
    if (index >= str.length()){
        cout<<str<<endl;
        return;
    }

    for (int j=index; j<str.length(); j++) {

        swap(str[index],str[j]);
        // recursive call
        stringPermutation (str, index+1);
        // backtracking
        swap (str[index], str[j]);
    }
}

int main() {
    string str = "odi";
    int index = 0;
    stringPermutation (str, index);
}
```

__/__/____

* Rat in a maze →

I/P → maze[3][3] = { {1, 1, 0},
{1, 1, 0},
{0, 1, 1} };

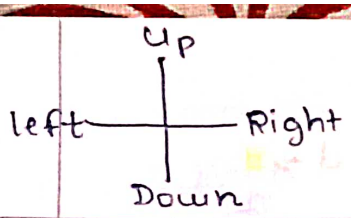
⇒ 0 indicates closed path & 1 indicates open path.
Source position = (0,0)
Destination position = (row-1, col-1)

O/P → DRDR
RDDR

Brief of code →

```
// this function will tell whether any position
// in maze is right position to reach destination
// or not

isSafe(maze, row, col, srcX, srcY, newX, newY, visited)
{
    if(
        1. Inside bound
        2. open path
        3. Not visited
      )
    return true;
  else
    return false;
}
```



These 4 movements are allowed only.

// this function gives the path to reach destination

```
printAllPath(maze, row, col, srcX, srcY, output, visited)
{
    // base case
    if (reached destination)
        print output and return

1. // Down → newX = srcX + 1 , newY = srcY
        if (isSafe){
          • mark visited
          • Push back 'D'. (D indicates Down)
          • recursive call
          • backtracking → pop back & unmark visited
        }
2. // right → newX = srcX , newY = srcY+1
        if (isSafe){
          • mark visited
          • Push back 'R' (R indicates right)
          • recursive call
          • backtracking → Pop back & unmark visited
        }
3. // left → newX = srcX , newY = srcY -1
        if (isSafe){
          • mark visited
          • Push back 'L' (L indicates left)
          • recursive call
          • backtracking → pop back & unmark visited
        }
4. // up → newX = srcX-1, newY = srcY
if(isSafe){ • mark visited • Push back 'U' (U indicates UP)
          • recursive call
          • backtracking → pop back & unmark visited }
```

Dry Run →

Maze →

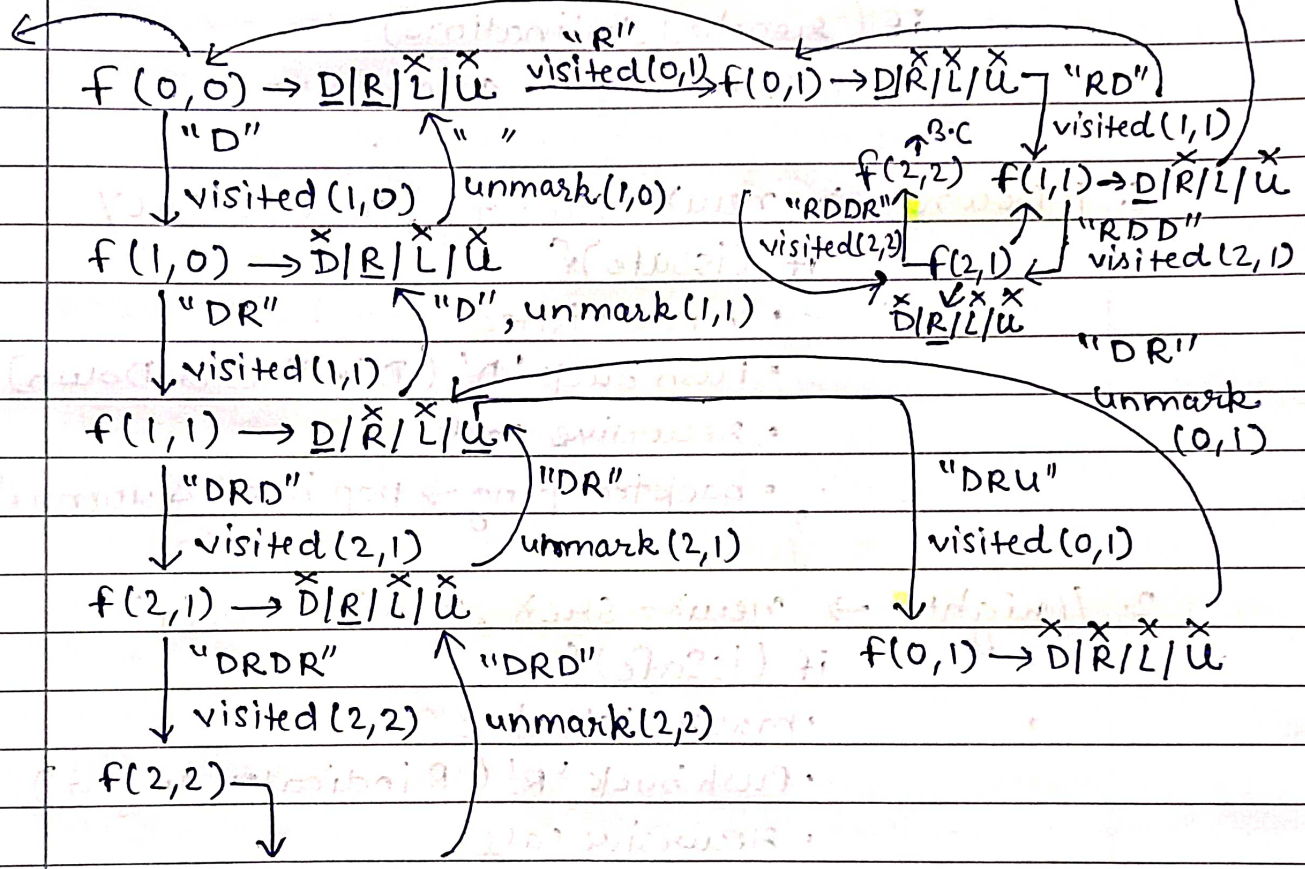
	0	1	2
0	1	1	0
1	1	1	0
2	0	1	1

Visited →

	0	1	2
0	FT	FTFT	F
1	FTF	FTFT	F
2	F	FTFTF	FTFTF

left k liye call jayegi which ultimately leads to no path

Source = (0,0) Destination = (row-1, col-1)



Output →

DRDR
RDDR

Code →

```
#include <bits/stdc++.h>
using namespace std;

bool isSafe (int maze[][3], int row, int col, int srcX,
int srcY, int newX, int newY, vector<vector<bool>> &visited)
{
    if (
        (newX >= 0 && newX < row) &&
        (newY >= 0 && newY < col) &&
          maze [newX][newY] == 1 &&
          visited[newX][newY] == false
        )
    {
      return true;
    }
    else{
        return false;
      }
}

void printAllPath (int maze[][3], int row, int col,
int srcX, int srcY, string &output, vector<vector< bool>>&visited)

    // base case
    // destination coordinates [row-1][col-1]
    if (srcX == row-1 && srcY == col-1){
        // destination reached
        cout << output << endl;
        return;
    }
```

// 1 case solve krlo baaki recursion sambhal lega

```
// down
  int newX = srcX+1;
  int newY = srcY;
if (issafe(maze, row, col, srcX, srcY, newX, newY, visited)){
        // mark visited
        visited[newX][newY]=true;
        output.push_back('D');
        // recursive call
printAllPath(maze, row, col, newX, newY; output, visited);
        // backtracking,
         output.pop_back();
         visited[newX][newY]=false;
    }

// right
     newX = srcX;
     newY = srcY+1;
if (issafe(maze, row, col, srcX, srcY, newX, newY, visited)){
        // mark visited
        visited[newX][newY]=true;
        output.push_back('R');
        // recursive call
printAllPath(maze, row, col, newX, newY, output, visited));
        // backtracking;
        output.pop_back();
        visited[newX][newY]=false;
    }
```

```
//left
    newX = srcX;
    newY = srcY-1;
if(isSafe(maze, row, col, srcX, srcY, newX, newY, visited)){
    //mark visited
    visited[newX][newY] = true;
    output.push_back('L');
    //recursive call
printAllPath(maze, row, col, newX, newY, output, visited);
    //backtracking
    output.pop_back();
    visited[newX][newY] = false;
  }

//up
    newX = srcX-1;
    newY = srcY;
if(isSafe(maze, row, col, srcX, srcY, newX, newY, visited)){
    //mark visited
    visited[newX][newY] = true;
    output.push_back('U');
    //recursive call
printAllPath(maze, row, col, newX, newY, output, visited);
    //backtracking
    output.pop_back();
    visited[newX][newY] = false;
  }
}
```

```
int main(){

    int maze[3][3]=
    {
        {1, 1, 0},
        {1, 1, 0},
        {0, 1, 1}
    };

    int row=3;
    int col =3;

    int srcX=0;
    int srcY=0;
    string output=" ";

// create a 2-D vector visited
vector<vector<bool>>visited(row,vector<bool>(col,0));

    if (maze[0][0]==0){
// source position is closed, it means rat cannot move
        cout<<"no path exists"<<endl;
    }
    else{
        visited[srcX][srcY]=true;
printAllPath(maze,row,col,srcX,srcY,output,visited);
    }
}
```